Quality Function Deployment (QFD): An Effective Technique For Requirements Acquisition

Tuyct-Lan Tran and Joseph S. Sherif
Jet Propulsion Laboratory, Software Assurance
California Institute of Technology, Pasadena, CA 91109 USA

Abstract

A general and accepted understanding of how to capture requirements, allocate or flow-down top-level requirements, verify and validate lower-level requirements, is at best sought in theory but not rigorously sought in practice. More often than not, the customers (or users) are blamed for not properly articulating their requirements or even understanding their own needs. However, the problem is deeper than that, and it involves not only the customers but also the system analysts or engineers, and designers as well.

This paper puts forward Quality Function Deployment (QFD) as an effective tool for the acquisition of requirements. QFD when applied to a project will: (1) improve communications between customers, system engineers, programmers and testers and thus contribute to project success, (2) enable alignment between customer requirements, product (or design) requirements, and cost requirements (or constraints), by explicitly correlating key product requirements to customer needs and expectations, (3) improve the management of requirements through rigorous prioritization, built-in traceability, and explicit tradeoff analysis, and (4) facilitate reengineering of some key processes or subprocesses, through focus on key performance requirements .

Introduction

Requirements engineering is one of the most crucial parts of the development process of any project, yet it is the least supported or least understood part due to the following reasons: (1) requirements are particularly difficult to specify and analyze since they may be derived from the needs of many different customers or people; (2) difficulty to achieve a complete understanding of the application domain within which the proposed system will function, as discussed by Rubenstein and Waters [6]; and (3) all relevant aspects of a proposed system may be difficult to capture in a single paradigm. This is due to the fact that each paradigm is embodied in a single requirement language that may have. its own limitations to express some important requirement.

The primary output of requirements engineering is a requirements specification that must be internally consistent; consistent with other existing documents; correct and complete with respect to satisfying users needs; understandable to users, designers and testers; and capable of serving as a basis for both design and test [4]. Hsia, et. al., [3] also assert that the quality of a product is only as good as the process that creates it; and that requirements engineering is one of the most crucial steps in this creation process. Hsia describes requirements engineering as the disciplined application of proven principles, methods, tools, and notations to map a proposed system's intended behavior and its associated constraints. This mapping includes: (1) identification and documentation of user needs, (2) development of a requirements document that describes how to satisfy user needs, (3) analysis and validation of the requirements document and (4) means to support the evolution of user needs.

Requirements Acquisition

'l'he principal problems in Requirements Acquisition include difficulties in: (1) agreement about requirements statements; (2) intra-team communication; (3) managing change, i.e., maintenance and evolution of initial requirements and identifying inconsistencies between initial and refined requirements; (4) formalism and abstraction in capturing objective reality, since constructed reality is, after all, a result of interactions among participants in the requirements process.

Curtis, et. al [2] identified two significant problems in requirements that may cause major difficulties during the development of projects: acquisition of accurate problem domain knowledge, and volatility of requirements. Any of these problems will contribute to low quality projects, budget overrun and schedule slip.

Lubars et. al., [5] assert that the traditional way of requirements capture by prose-like unstructured, obscure and somewhat ambiguous statements is no longer effective; and they recommend that new techniques and tools for requirements engineering should be used.

Tran et. al., [7] describe successful projects as those that meet valid functional requirements as well as users expectations; adhere to the spirit of process methods that promote rigor, discipline and continuous refinement; and are accomplished on time and within budget. They assert that among the key attributes exhibited by successful projects are the constant visibility of requirements, and the commitment of sponsors as well as users to this same set of governing requirements.

Quality Function Deployment (QFD)

To date, Quality Function Deployment (QFD) is the only technique discovered by the authors that facilitates the concurrent capture of problem domain knowledge and solution domain knowledge without requiring formalism, and yet facilitates requirements validation -- requirements validation from the customer's perspective..

Yoji Akao introduced QFD to the United States in October 1983 in a short article in the journal Quality Promess [8].

The goal of QFD is to deploy the "voice of the customer" (VOC) throughout the product's entire technical specifications and resource requirements. This VOC is intended to represent the customer's viewpoint of the customer's problem or need. Detailed matrices listing the customer's rat ed "whats" (or expectations) are correlated with the "hews" , to show how each customer requirement will be met, and which team(s) will be responsible for each performance component [1]. This systematic technique of listening to the voice of the customer, and ensuring the traceability of product design (or, solution domain) to the customer's requirements are the most crucial aspects of QFD in delivering high quality products.

The customer-requirements planning matrix is the most important element of the QFD implementation. Customer requirements (or, customer attributes) are customer needs in customer's terms and language. The technical features are the de sign attributes expressed in the language of the system engineer, designer, and developer. These features must be measurable, since the output will be controlled and compared to objective targets. Relating the customer attributes (or customer requirements) to the technical features (also referred to as product characteristics or performance requirements throughout this paper) will show the strength of the relationship between them; and show whether the attributes are addressed fully and properly or whether the final product will have difficulty in meeting customer needs.

QFD and Software Development

Since 1980 several companies in Japan, Europe and the United States have used Total Quality Management (l'QM), concurrent engineering and QFD techniques for software development, at least for the first phases of software development (software requirements and specifications).

More specifically, CSK of Japan has been using QFD for software since 1985. Figure 1 shows the steps in their QFD activities for developing the company's software [7]. These steps include: (1) collecting customer requirements (from original interview data and brainstorming sessions by a cross-functional team); (2) generating the quality requirements (by identifying the several levels of product characteristics that correlate with the customer requirements (or "the demanded quality"); (3) generating the function-based requirements (by exploding the system functions into several levels of functional requirements); (4) establishing the planned quality.

This fourth step consists of: (a) extracting and analyzing selected parameters from the quality requirements, (b) deciding which parameters are most strongly correlated with the demanded quality and become the product's "quality characteristics", (c) establishing a standard value for each quality characteristic (also referred to as technical feature, or product characteristic, or performance requirement, throughout this paper), (d) deploying these quality characteristics into processes, and (e) implementing these processes in software development.

CSK's next major activities of the QFD technique arc as follows: (5) analyzing the relationships between the impact of the i mplemented soft ware on customer demands (or customer requirements) and the quality characteristics; (6) capturing the results of this evaluation (by rating customer satisfaction for each customer requi rernent or demand); (7) analyzing the relationships between the deployed software processes and the selected quality characteristics; (8) refining the planned-quality chart, for the next development effort. At present, CSK is developing a QFD support system using artificial intelligence for improving the company's software development activities, efforts, and productivity.

Drawing from the QFD work performed by the Software Assurance engineers at JPL, which includes some adaptations of CSK's concepts, the authors believe that developing a QFD support system that integrates software techniques such as data flow diagramming and object-oriented development wi ll be key to dramatically improving JPL software development activities, and to delivering quality products (i.e., products that meet customer needs, within schedule and budget).

Discussion of QFD Benefits

QFD has become an effective and important investment for many companies because it is the cornerstone for implementing concurrent engineering and Total Quality Management (TQM). In support of TQM's goal for maintaining or improving quality, cost, procedures and systems, the QFD technique, indeed, provides an explicit mechanism for capturing and incorporating the voice of the customer early into the production process, whether at pre-project, conceptual design, or high-level design phase; i.e., into the front -end of the development lifecycle where that voice should be the sharpest.

QFD can also be part of business reengineering, in promoting radical business improvement. In many software-intensive project environments, while the concept of customer focus varies from ceremonial attention being paid to it, to having some customer representatives participate on review boards, it is often done from the engineering viewpoint, rather than from the customer viewpoint. QFD promotes the outside-in approach, rather than the inside-out approach, as it attempts to identify the value-adding features of the software-intensive product. As QFD assigns priorities or weights to product features, some of these could represent radical improvements for both the product and the associated process. When reengineering could be accomplished through focusing on essential design parameters, and by concentrating on those that link back to the customer's true needs, reengineering would be less intrusive to organizations and more likely to succeed.

With respect to customer fulfillment, the benefits of QFD application to software projects are that: (1) customer needs are integrated into product design upfront, and without generating a lengthy or unrealistic customer requirements document; (2) product requirements can be better fine-tuned via iterative specification of performance characteristics and of the relationships between these performance characteristics and the customer needs; (3) all product-requirements specified are measurable and testable.

With respect to radical changes in software development-process, the advantage of QFD is that it eliminates the typical productivity drains associated with requirements management. (1) The capture of requirements is tremendously more cost-effective, because it is faster and because of more accurate customer requirements and product requirements. Faster and more accurate customer requirements, by including an experienced customer or strong customer advocate in the QFD team. Faster and more accurate product requirements, by including in the QFD team product designers and technologists who can listen to customers. The mapping of product requirements to customer requirements is more consistent in degree. of expressiveness through quantified relationships. Consequently, errors in requirements capture, requirements analysis and design are fewer, and there are fewer design changes late in development or production -- which in turn reduces overall product cycle time and project development costs. (2) Cost planning becomes more specifically tied to product features, and in explicit ways. The allocation of performance requirements to one or more partitions of the overall product provides the necessary link to the generation and costing of a product-oriented work breakdown structure (WBS). (3) The technical decision-making process becomes more explicit, thereby contributing to more participation from all development team members as well as to better team focus. Tradeoff analysis, indeed, is more explicit and concentrated on specific, potentially-conflicting design features or bottlenecks. As control points are clarified, consensus-building becomes easier, and an informed balance between quality and cost is made. (4) The documenting and tracing of requirements become side-products of analysis and design, and do not generate reams of post-facto, error prone, and expensive documents. (5) Finally, the coupling of higher -integrity requirements (discussed above.) with a more explicit technical decision-making process, and with minimum documentation-related distractions, enables the project to avoid problems, such as erroneous or untestable requirements, requirements volatility, and gold plating.

It has been reported that although only 6 percent of project cost and 10 percent of project duration are spent in the requirements phase, it costs about 10 times more to repair a defect during implementation than during the requirements phase, and it costs between 100 and 200 times more during maintenance. Historical project-performance records also show that 30-to-50% of the cost of building a hardware-software system are spent in finding and correcting defects. For certain application areas, about 60-90% of software failures observed are said not to be caused by code errors, but are attributed to requirements errors. These requirements-related problems, when coupled with the Total Quality Management (TQM) goals of increased product quality and lowered cost, suggest that the area for highest-return on quality investment is in the prevention of defects with greatest-impact (or greatest amplification-rate) potential; i.e., at the requirements level and at the front-end of the development life-cycle. This front-end could mean preproject phase, prototyping phase, exploratory or conceptual development phase.

QFD targets both the front-end of the development process and the product life-cycle itself, for improvement (either small or dramatic). By simultaneously capturing customer requirements, product requirements and the results of rigorous analysis from a knowledgeable team, the set of QFD charts becomes the repository for product plans and specifications. This repository constitutes the single source for configuration control and requirements visibility, for use and referencing whether by management, by development team, or by customers.

Lastly, an important contribution of QFD to people management, less evident in terms of cost savings, is that it acts as a powerful catalyst for team building and for infusing technical excitement into the consensus building process. This “soft” benefit becomes sharper and more critical, when one considers that the QFD team could be the reengineering team or vice-versa.

Conclusion

QFD is an effective and promising technique in alleviating the problems associated with the early phases of requirements and specifications. From the TQM perspective, QFD is an excellent avenue for specification of the “right product” at the right price. QFD, indeed, is a cross-functional tool that enables organizations to focus on key customer demands and develop appropriate responses to those needs. When these responses involve dramatic changes in some aspects of product performance, QFD also becomes a technique for reengineering. Most critically, QFD systematizes rigor in requirements activities, while maintaining relatively low documentation and requirements-tracing costs. Lastly, the authors would like to conclude that, although QFD has been accepted as a useful tool for product planning, its most unique potential as a catalyst for parameter design of information systems can be fully realized only after a QFD support system is put in place, which integrates software development methodologies.

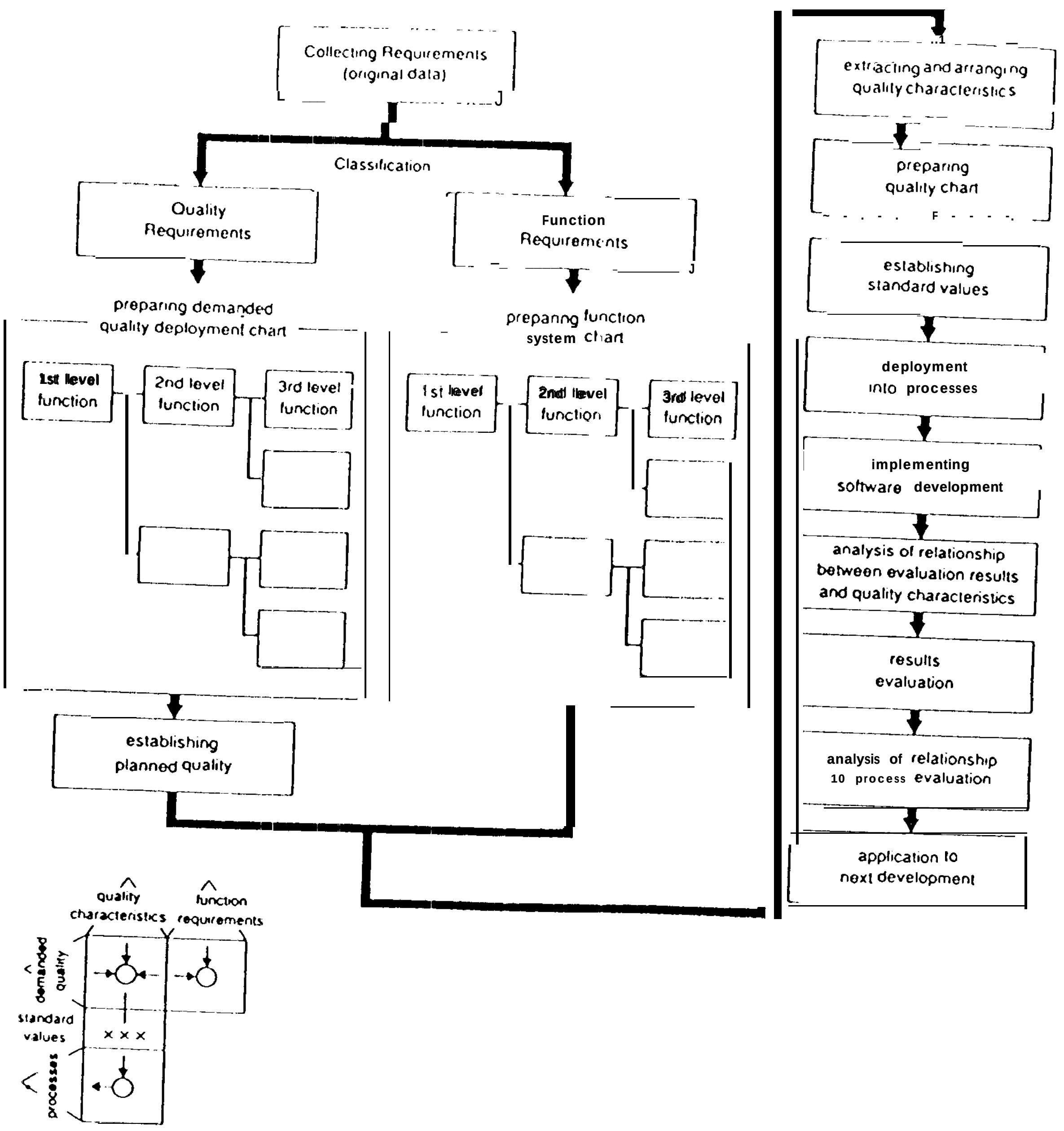
Collecting Requirements (original data)
Classification
Quality Requirements
Function Requirements
preparing demanded quality deployment chart
preparing function system chart
1st level function
2nd level function
3rd level function
establishing planned quality
quality characteristics
function requirements
demanded quality
standard values
processes
extracting and arranging quality characteristics
preparing quality chart
establishing standard values
deployment into processes
implementing software development
analysis of relationship between evaluation results and quality characteristics
results evaluation
analysis of relationship to process evaluation
application to next development

Figure 1. Steps in QFD Activities

REFERENCES

[1] Clemmer, J. and B. Sheeby, Firing on All Cylinders, Irwin, Homewood, Ill, 1992

[2] Curtis, B., Krasner, H., and N. Iscoe, "A Field Study of The Software Design Process for Large Systems," Comm. ACM, 31, 11, pp. 1268-1287, 1988.

[3] Hsia, P., and A, T. Yaung, "Another Approach to System Decomposition: Requirements Clustering," Proc. The Twelfth Annual Int'l Computer Software and Applications Conference (COMPSAC), Chicago, ILL, pp. 75-82, 1988.

[4] Hsia, P., Jayara, Jan, S., Gao, J., King, D., Toyoshima, Y., and C. Chen, "Formal Approaches to Scenario Analysis," IEEE Software 11, 2, pp. 33-41,1994.

[5] Lubars, M., Potts, C., and C. Richter, "A Review of The State of The Practice in Requirements Modeling," Proc. IEEE Int. Symp. on Req. Eng., pp. 2-14, San Diego, CA, January 4-6, 1993.

[6] Reubenstein, H. B., and R. C. Waters, "The Requirements Apprentice: Automated Assistance For Requirements Engineering," IEEE Trans. on SE, 17, 3, pp. 226 - 240, 1991.

[7] Tran, T. L., Lee, S., and J. S. Sherif, "The Network Operations Control Center (NOCC) Upgrade Task: Lessons Learned," TDA Progress Report Number 42-118, NASA, JPL, pp. 160-168, 1994.

[8] Yoshizawa, T., Togari, H., and T. Koribayashi, "QFD, Integrating Customer Requirements into Product Design, (Akao, Y. editor), Productivity Press, Cambridge, MA. 1990.